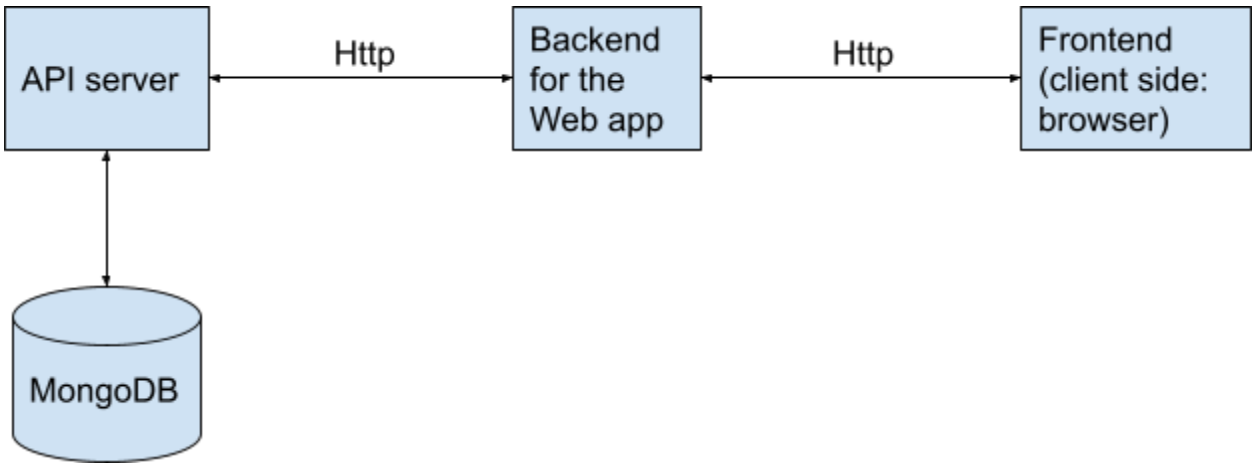
Current Architecture/design decision:



Reason for using MongoDB: I picked mongoDB because it's schemaless. Instead of MongoDB If we use any SQL DB such as SQLite3, then we need to think about the table schema. Also, I need to create a separate script for table creation. Due to use of mongoDB I didn't need to create any tables separately. MongoDb creates a collection when it gets the first data/record. It's good for rapid prototyping.

Reason for using a Backend for the web app: Obviously we can use Javascript to directly call the API endpoints from frontend. However, having a separate backend for the application means we can make API calls with an API key in the header. Also, before reaching API server, we can do user authentication at the application backend.

Creation of Swagger file: Wrote the Swagger file first. It helps me to design the API interfaces/endpoints first. Later endpoint development became easier.

Use of Docker and Docker-compose makes it easy to run all 3 services in a single command. Changing code in the source repo also updates the running application immediately which makes my development faster. Use of Docker file will also make sure the portability of this application/api-service.

Future Iterations:

- Improvement in API service endpoints:
    - Using API key for security
    - Use HTTPS (Nginx web server in front of the web application)
    - Versioning through url such as '/v1/forecast'
    - Use SQL DB instead of NoSQL. I think the datetime range search would be faster with SQL databases.

    - Currently the post API call has a maximum limit of 1000 records. We can remove this restriction and convert this to an asynchronous API call. Need to use distributed task queueing and massage broker (celery + RabbitMQ)
    - Need sanitization/validation of query parameters
    - Currently the API is strictly considering 7 days (weekly) of forecasts. We can parametrize this as N days of forecasts.
    - Current implementation inserts each record one by one. We should modify the code to do bulk insertion in the database. This will speed up the api performance.
    - Use python Logger.
    - Use log tables for audit purposes.
- Improvement in Frontend:
    - Use a Frontend framework/library such as React, Angular or Vue to make the web page responsive and faster.
    - Complete the display of the weekly forecasting. Use of Google charts might be a good option to display time series data (temperature, wind speed and wind direction) when specific time information is missing/not-provided.
    - Form data validation
    - Use date-time picker

Conclusion:

- The provided code for the task is fully runnable.
- However, the frontend is very simple or incomplete. Need to use/develop appropriate UI components to make it more professional.

Thank you.